1

Welcome to Visual Basic 2008

COPYRIGHTED MATERIAL

This is an exciting time to enter the world of programming with Visual Basic 2008 and Windows Vista. Windows Vista represents the first Windows operating system upgrade since Windows XP was first released in 2002. A lot has changed in the Windows user interface and Visual Basic 2008 makes it easy to write professional-looking Windows applications as well as web applications and web services. Haven't upgraded to Windows Vista yet? No worries, Visual Basic 2008 also allows you to write professional-looking applications for Windows XP as well.

The goal of this book is to help you use the Visual Basic 2008 programming language, even if you have never programmed before. You will start slowly and build on what you have learned in subsequent chapters. So take a deep breath, let it out slowly, and tell yourself you can do this. No sweat! No kidding!

Programming a computer is a lot like teaching a child to tie his shoes. Until you find the correct way of giving the instructions, not much is accomplished. Visual Basic 2008 is a language you can use to tell your computer how to do things. But, like a child, the computer will understand only if you explain things very clearly. If you have never programmed before, this sounds like an arduous task, and sometimes it can be. However, Visual Basic 2008 gives you an easy-to-use language to explain some complex tasks. Although it never hurts to have an understanding of what is happening at the lowest levels, Visual Basic 2008 frees the programmer from having to deal with the mundane complexities of writing Windows applications. You are free to concentrate on solving real problems.

Visual Basic 2008 helps you create solutions that run on the Microsoft Windows operating systems, such as Windows Vista, Windows Server 2008, and Windows Mobile 6. If you are looking at this book, you might have already felt the need or desire to create such programs. Even if you have never written a computer program before, as you progress through the Try It Out exercises in this book, you will become familiar with the various aspects of the Visual Basic 2008 language, as well as its foundations in the Microsoft .NET Framework. You will find that it is not nearly as difficult as you had imagined. Before you know it, you will feel quite comfortable creating a variety of different types of programs with Visual Basic 2008.

Visual Basic 2008 can also be used to create web applications and web services as well as mobile applications that can run on Pocket PCs or SmartPhones. However, you will begin by focusing on Windows applications before extending your boundaries to other platforms.

This chapter covers the following topics:

- Event-driven programming
- The installation of Visual Basic 2008
- A tour of the Visual Basic 2008 Integrated Development Environment (IDE)
- How to create a simple Windows program
- How to use the integrated Help system

Event-Driven Programming

A Windows program is quite different from yesteryear's MS-DOS program. A DOS program follows a relatively strict path from beginning to end. Although this does not necessarily limit the functionality of the program, it does limit the road the user has to take to get to it. A DOS program is like walking down a hallway; to get to the end you have to walk down the hallway, passing any obstacles that you may encounter. A DOS program would only let you open certain doors along your stroll.

Windows, on the other hand, opened up the world of *event-driven programming*. *Events* in this context include clicking a button, resizing a window, or changing an entry in a text box. The code that you write responds to these events. In terms of the hallway analogy: In a Windows program, to get to the end of the hall, you just click the end of the hall. The hallway can be ignored. If you get to the end and realize that is not where you wanted to be, you can just set off for the new destination without returning to your starting point. The program reacts to your movements and takes the necessary actions to complete your desired tasks.

Another big advantage in a Windows program is the *abstraction of the hardware*; which means that Windows takes care of communicating with the hardware for you. You do not need to know the inner workings of every laser printer on the market just to create output. You do not need to study the schematics for graphics cards to write your game. Windows wraps up this functionality by providing generic routines that communicate with the drivers written by hardware manufacturers. This is probably the main reason that Windows has been so successful. The generic routines are referred to as the Windows *application programming interface* (API), and the classes in the .NET Framework take care of communicating with those APIs.

Before Visual Basic 1.0 was introduced to the world in 1991, developers had to be well versed in C and C++ programming, as well as the building blocks of the Windows system itself, the Windows API. This complexity meant that only dedicated and properly trained individuals were capable of turning out software that could run on Windows. Visual Basic changed all of that, and it has been estimated that there are now as many lines of production code written in Visual Basic as in any other language.

Visual Basic changed the face of Windows programming by removing the complex burden of writing code for the user interface (UI). By allowing programmers to *draw* their own UI, it freed them to concentrate on the business problems they were trying to solve. When the UI is drawn, the programmer can then add the code to react to events.

Visual Basic has also been *extensible* from the very beginning. Third-party vendors quickly saw the market for reusable modules to aid developers. These modules, or *controls*, were originally referred to as VBXs (named after their file extension). Prior to Visual Basic 5.0, if you did not like the way a button behaved, you could either buy or create your own, but those controls had to be written in C or C++. Database access utilities were some of the first controls available. Version 5 of Visual Basic introduced the concept of *ActiveX*, which allowed developers to create their own *ActiveX controls*.

When Microsoft introduced Visual Basic 3.0, the programming world changed significantly. Now you could build database applications directly accessible to users (so-called *front-end applications*) completely with Visual Basic. There was no need to rely on third-party controls. Microsoft accomplished this task with the introduction of *Data Access Objects* (DAO), which allowed programmers to manipulate data with the same ease as manipulating the user interface.

Versions 4.0 and 5.0 extended the capabilities of Version 3.0 to allow developers to target the new Windows 95 platform. They also made it easier for developers to write code, which could then be manipulated to make it usable to other language developers. Version 6.0 provided a new way to access databases with the integration of *ActiveX Data Objects* (ADO). The ADO feature was developed by Microsoft to aid web developers using Active Server Pages (ASP) to access databases. All of the improvements to Visual Basic over the years have ensured its dominant place in the programming world — it helps developers write robust and maintainable applications in record time.

With the release of Visual Basic .NET in February 2002, most of the restrictions that used to exist have been obliterated. In the past, Visual Basic was criticized and maligned as a "toy" language, because it did not provide all of the features of more sophisticated languages such as C++ and Java. Now, Microsoft has removed these restrictions and made Visual Basic .NET a very powerful development tool. This trend has continued with the release of Visual Basic 2003, Visual Basic 2005, and the latest release, Visual Basic 2008. Each new release of the Visual Basic .NET programming language brings about many new trends, features, and improvements, making it a great choice for programmers of all levels.

Installing Visual Basic 2008

You may own Visual Basic 2008 in one of the following forms:

- As part of Visual Studio 2008, a suite of tools and languages that also includes C# (pronounced *C-sharp*) and Visual C++. The Visual Studio 2008 product line includes Visual Studio Professional Edition or Visual Studio Tools Team Editions. The Team Edition versions come with progressively more tools for building and managing the development of larger, enterprise-wide applications.
- As Visual Basic 2008 Express Edition, which includes the Visual Basic 2008 language, and a reduced set of the tools and features that are available with Visual Studio 2008.

Both of these products enable you to create your own applications for the Windows platform. The installation procedure is straightforward. In fact, the Visual Studio Installer is smart enough to figure out exactly what your computer requires to make it work.

The descriptions in the following Try It Out exercise are based on installing Visual Studio 2008 Professional Edition. Most of the installation processes are straightforward, and you can accept the

default installation options for most environments. So, regardless of which edition you are installing, the installation process should be smooth when accepting the default installation options.

Try It Out Installing Visual Basic 2008

1. The Visual Studio 2008 DVD has an auto-run feature, but if the Setup screen does not appear after inserting the DVD, you need to run Setup.exe from the root directory of the DVD. To do this, click the Windows Start menu at the bottom left of your screen and then select the Run start menu item or browse to the Setup program on the DVD. In the Run dialog box, you can click the Browse button to locate the `setup.exe` program on your DVD. Then click the OK button in the Run dialog box to start the setup program. After the setup program initializes, you will see the initial screen as shown in Figure 1-1.

Visual Studio 2008 Setup
Microsoft
Visual Studio 2008 Setup
Install Visual Studio 2008
Install Visual Studio 2008 features and required components.
msdn
Install Product Documentation
Install the MSDN Library, which includes Help for Visual Studio.
Check for Service Releases
Check for the latest Service Releases to ensure optimal functionality of Visual Studio 2008.
View ReadMe
Exit

Figure 1-1

2. The dialog box shown in Figure 1-1 shows the order in which the installation will occur. To function properly, Visual Studio 2008 requires various updates to be installed depending on the operating system that you have (for example, Service Pack 2 on Windows XP). The setup program will automatically inform you of these updates if they are not installed. You should install those updates first and then return to the Visual Studio 2008 setup program. The individual updates required are different from the service releases listed as the third option in Figure 1-1. Step 1 of the setup program will install Visual Studio 2008 so click the Install Visual Studio 2008 link shown in Figure 1-1.

3. The next step in the installation process asks you if you want to send the setup information from the installation of Visual Studio 2008 to Microsoft. This is a good idea to help streamline the installation process of future editions of Visual Studio, and no personal information will be sent. You can click the Next button at this screen after you have selected or cleared the check box indicating whether or not you want this information sent.

4. The third step in the installation process is the license agreement. Read the license agreement and then select the option button indicating your acceptance of the licensing terms. Then click the Next button to continue.

5. As with most setup programs, you are presented with a choice of options to be installed as shown in Figure 1-2. The default installation installs the recommended product features as determined by Microsoft. You have the option to choose the default installation, a full installation, or to customize the installation. When choosing the custom installation feature, you will be presented with a dialog box allowing you to choose the languages and features of each language to be installed. If disk space allows, it is recommended that you choose a full installation. However, if you choose to customize the installation and omit some features from being installed, you can always install those features later by rerunning the setup program. After choosing your installation option, click the Install button to have those features installed.

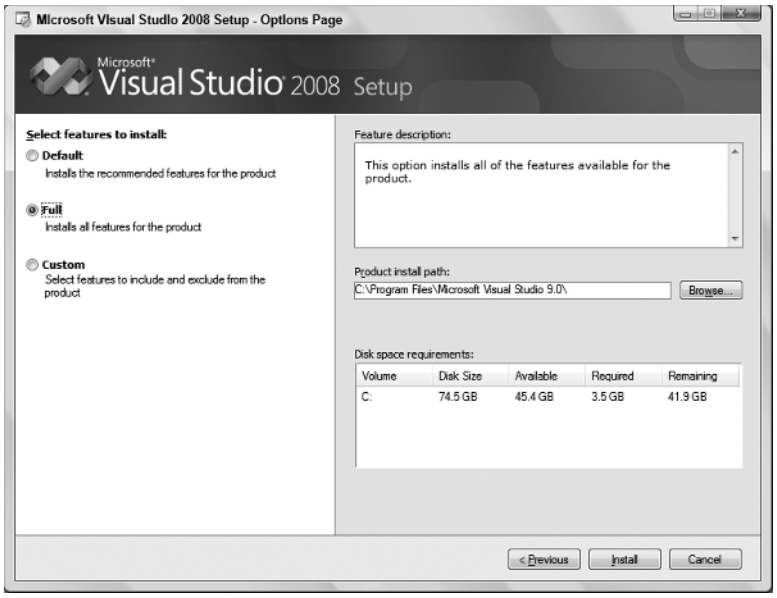

Figure 1-2

6. The first component that is installed is the Microsoft .NET Framework version 3.5. During the installation of this component you will be required to restart your computer. After your computer has restarted and you log back in, the setup program will continue. Note to Windows Vista users: you will be prompted that the setup program needs to run and will need to grant permission to let the setup program continue. After the setup program continues, you can sit back and relax while all of the features are being installed. The setup program can take anywhere from 20 minutes on up depending on the installation features chosen and the speed of your computer.

7. Once the installation has been completed, you will be presented with a dialog box informing you of the status of the installation. Here you can see any problems that the setup program encountered. At this point you are encouraged to update your computer with the latest security patches and a link is provided in the notes to Windows Update. When you have finished reviewing the setup status, click the Finish button to move on to the next step.

8. If you chose to have your setup information sent to Microsoft, the next step will be a dialog box sending the setup information. This dialog box requires no action on your part and it will automatically close when finished. The next dialog box is the one shown earlier in Figure 1-1 with the option to install the production documentation enabled. Click the Install Product Documentation link to install the MSDN library.

9. The first step in installing the MSDN library is choosing whether to send the setup information to Microsoft. Make the appropriate choice and then click the Next button to continue. Again, it is recommended to send this information to help streamline future MSDN library installations.

10. Next, read and accept the license agreement. After you click the option button to accept the license agreement, click the Next button to continue.

11. Like the installation of Visual Studio 2008, the MSDN library installation provides you with the options to choose the installation that best suits your needs, as shown in Figure 1-3. If you chose to install the complete Visual Studio 2008 product set then you'll most likely want to choose the full installation of the MSDN library. After making your installation option choice, click the Install button to begin the installation.

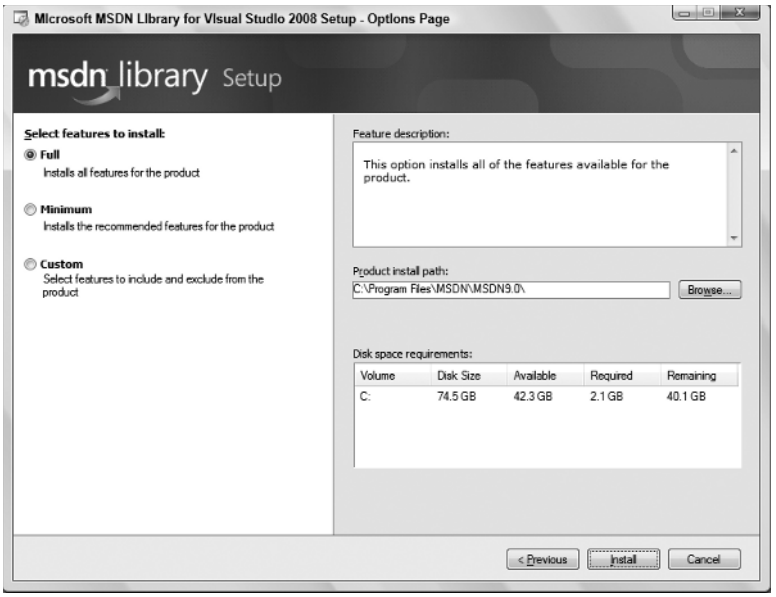

Figure 1-3

If you have the spare hard drive space, it is a very good idea to install the full documentation. That way you have access to the full library, which will be important if you choose a limited set of options during the install and later add more features.

12. After the MSDN documentation has been installed, you are presented with a dialog box informing you of the status of the installation. Click the Finish button to be returned to the initial setup screen again. The Check for Service Releases option is now available.

It is a good idea to select Service Releases to check for updates. Microsoft has done a good job of making software updates available through the Internet. These updates can include anything from additional documentation to bug fixes. You will be given the choice to install any updates through a Service Pack CD or the Internet. Obviously, the Internet option requires an active connection. Since updates can be quite large, a fast connection is highly recommended.

After you have performed the update process, Visual Studio 2008 is ready to use. Now the real fun can begin! So get comfortable, relax, and enter the world of Visual Basic 2008.

The Visual Basic 2008 IDE

You don't need Visual Basic 2008 to write applications in the Visual Basic .NET language. The ability to run Visual Basic .NET code is included with the .NET Framework. You could write all of your Visual Basic .NET code using a text editor such as Notepad. You could also hammer nails using your shoe as a hammer, but that slick pneumatic nailer sitting there is a lot more efficient. In the same way, by far the easiest way to write in Visual Basic .NET code is by using the Visual Studio 2008 IDE. This is what you see when working with Visual Basic 2008 — the windows, boxes, and so on. The IDE provides a wealth of features unavailable in ordinary text editors — such as code checking, visual representations of the finished application, and an explorer that displays all of the files that make up your project.

The Profile Setup Page

An IDE is a way of bringing together a suite of tools that makes developing software a lot easier. Fire up Visual Studio 2008 and see what you've got. If you used the default installation, go to your Windows Start menu and then select All Programs ⇨ Microsoft Visual Studio 2008 ⇨ Microsoft Visual Studio 2008. A splash screen will briefly appear, and then you see the Choose Default Environment Settings dialog box. Select the Visual Basic Development Settings option and click Start Visual Studio. After Visual Studio configures the environment based on the chosen settings, the Microsoft Development Environment will appear, as shown in Figure 1-4.

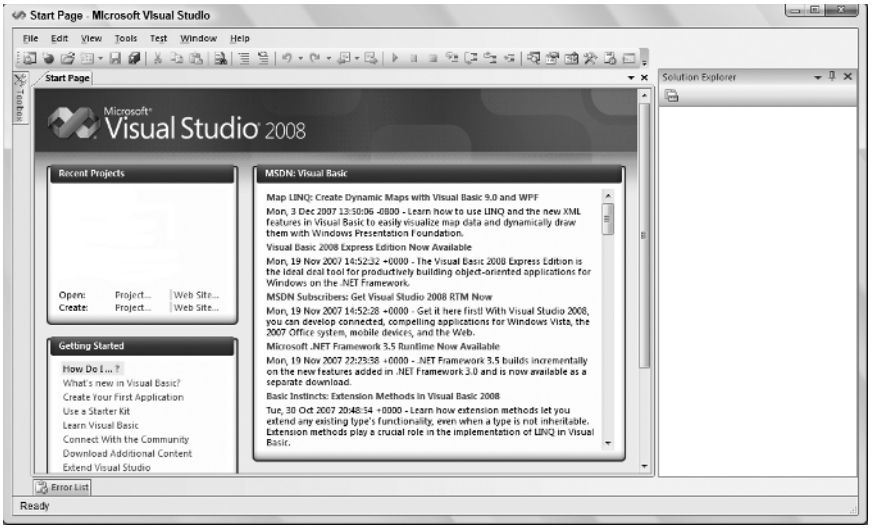
Start Page - Microsoft Visual Studio
File Edit View Tools Test Window Help
Start Page
Solution Explorer
Microsoft Visual Studio 2008
Recent Projects
Open: Project... Web Site...
Create: Project... Web Site...
MSDN: Visual Basic
Map LINQ: Create Dynamic Maps with Visual Basic 9.0 and WPF
Visual Basic 2008 Express Edition Now Available
MSDN Subscribers: Get Visual Studio 2008 RTM Now
Microsoft .NET Framework 3.5 Runtime Now Available
Basic Instincts: Extension Methods in Visual Basic 2008
Getting Started
How Do I ... ?
What's new in Visual Basic?
Create Your First Application
Use a Starter Kit
Learn Visual Basic
Connect With the Community
Download Additional Content
Extend Visual Studio
Error List
Ready

Figure 1-4

The Menu

By now, you may be eager to start writing some code. Begin your exploration of the IDE by looking at the menu and toolbar, which are not really all that different from the toolbars and menus you have seen in Microsoft Office 2003 (although they differ from the ribbon bars in Microsoft Office 2007).

The Visual Studio 2008 menu is *dynamic*, which means items will be added or removed depending on what you are trying to do. When looking at the blank IDE, the menu bar consists only of the File, Edit, View, Tools, Window, and Help menus. When you start working on a project, however, the full Visual Studio 2008 menu appears as shown in Figure 1-5.

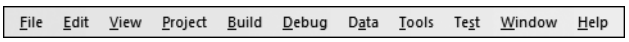

Figure 1-5

At this point, there is no need to cover each menu topic in detail. You will become familiar with each of them as you progress through the book. Here is a quick rundown of what activities each menu item pertains to:

- **File:** Most software programs have a File menu. It has become the standard where you should find, if nothing else, a way to exit the application. In this case, you can also find ways of opening and closing single files and whole projects.
- **Edit:** The Edit menu provides access to the common items you would expect: Undo, Redo, Cut, Copy, Paste, and Delete.
- **View:** The View menu provides quick access to the windows that exist in the IDE, such as the Solution Explorer, Properties window, Output window, Toolbox, and so on.

- **Project:** The Project menu allows you to add various files to your application such as forms and classes.
- **Build:** The Build menu becomes important when you have completed your application and want to run it without the use of the Visual Basic 2008 environment (perhaps running it directly from your Windows Start menu, as you would any other application such as Word or Access).
- **Debug:** The Debug menu allows you to start and stop running your application within the Visual Basic 2008 IDE. It also gives you access to the Visual Studio 2008 debugger. The debugger allows you to step through your code while it is running to see how it is behaving.
- **Data:** The Data menu enables you to use information that comes from a database. It allows you to view and add data sources, and preview data. Chapters 16 and 17 will introduce you to working with databases.
- **Tools:** The Tools menu has commands to configure the Visual Studio 2008 IDE, as well as links to other external tools that may have been installed.
- **Test:** The Test menu provides options that allow you to create and view unit tests for your application to exercise the source code in various scenarios.
- **Window:** The Window menu has become standard for any application that allows more than one window to be open at a time, such as Word or Excel. The commands on this menu allow you to switch between the windows in the IDE.
- **Help:** The Help menu provides access to the Visual Studio 2008 documentation. There are many different ways to access this information (for example, through the help contents, an index, or a search). The Help menu also has options that connect to the Microsoft web site to obtain updates or report problems.

The Toolbars

Many toolbars are available within the IDE, including Formatting, Image Editor, and Text Editor, which you can add to and remove from the IDE through the View ⇨ Toolbars menu option. Each one provides quick access to often-used commands, preventing you from having to navigate through a series of menu options. For example, the leftmost icon (New Project) on the default toolbar (called the Standard toolbar), shown in Figure 1-6, is available from the menu by navigating to File ⇨ New ⇨ Project.

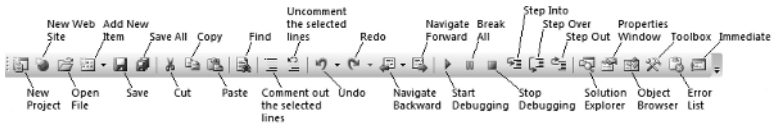

Figure 1-6

The toolbar is segmented into groups of related options, which are separated by vertical bars. The first six icons provide access to the commonly used project and file manipulation options available through the File and Project menus, such as opening and saving files.

The next group of icons is for editing (Cut, Copy, and Paste). The next icon is for finding and replacing items in your code.

The third group of icons is used for commenting out and un-commenting sections of code. This can be useful in debugging when you want to comment out a section of code to determine what results the program might produce by not executing those lines of code.

The fourth group of icons is for undoing and redoing edits and for navigating through your code.

The fifth group of icons provides the ability to start (via the green triangle), pause, and stop your application. You can also use the last three icons in this group to step into your code line by line, step over entire sections of code, and step out of a procedure. These icons will be covered in depth in Chapter 10.

The final group of icons provides quick links to the Solution Explorer, Properties window, Object Browser, Toolbox, Error List, and the Immediate window. If any of these windows is closed, clicking the appropriate icon will bring it back into view.

> *If you forget what a particular icon does, you can hover your mouse pointer over it so that a tooltip appears displaying the name of the toolbar option.*

You could continue to look at each of the windows in the IDE by clicking on the View menu and choosing the appropriate window. But, as you can see, they are all empty at this stage and therefore not too revealing. The best way to look at the capabilities of the IDE is to use it while writing some code.

Creating a Simple Application

To finish your exploration of the IDE, you need to create a project, so that the windows shown earlier in Figure 1-4 have some interesting content for you to look at. In the following Try It Out exercise, you are going to create a very simple application called HelloUser that will allow you to enter a person's name and display a greeting to that person in a message box.

Try It Out — Creating a HelloUser Project

1. Click the New Project button on the toolbar.

2. In the the New Project dialog box, select Visual Basic in the Project Types tree-view box to the left and then select Windows beneath it. The Templates box on the right will display all of the available templates for the project type chosen. Select the Windows Forms Application template. Finally, type **Hello User** in the Name text box and click OK. Your New Project dialog box should look like Figure 1-7.

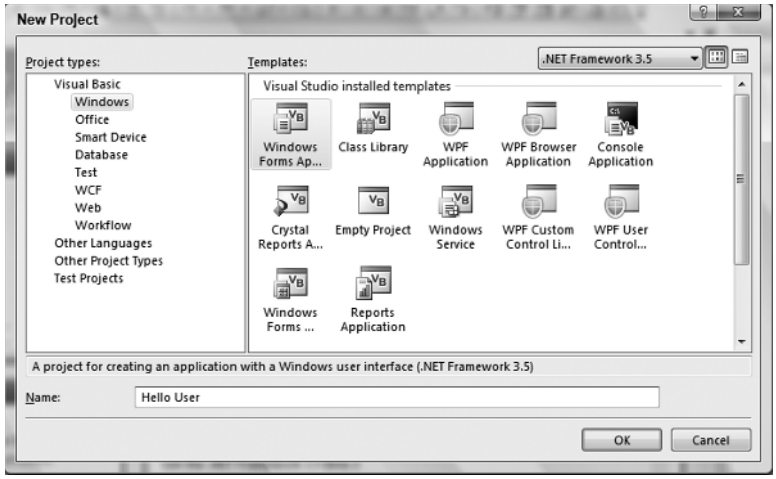

Figure 1-7

Visual Studio 2008 allows you to target your application to a specific version of the Microsoft .NET Framework. The combo box in the upper right corner of the New Project dialog box has version 3.5 selected, but you can target your application to version 3.0 or even version 2.0 of the .NET Framework.

The IDE will then create an empty Windows application for you. So far, your Hello User program consists of one blank window, called a Windows Form (or sometimes just a form), with the default name of `Form1.vb`, as shown in Figure 1-8.

Whenever Visual Studio 2008 creates a new file, either as part of the project creation process or when you create a new file, it will use a name that describes what it is (in this case, a form) followed by a number.

Windows in the Visual Studio 2008 IDE

At this point, you can see that the various windows in the IDE are beginning to show their purposes, and you should take a brief look at them now before you come back to the Try It Out exercise. Note that if any of these windows are not visible on your screen, you can use the View menu to show them. Also, if you do not like the location of any particular window, you can move it by clicking its title bar (the blue bar at

the top) and dragging it to a new location. The windows in the IDE can *float* (stand out on their own) or be *docked* (as they appear in Figure 1-8). The following list introduces the most common windows:

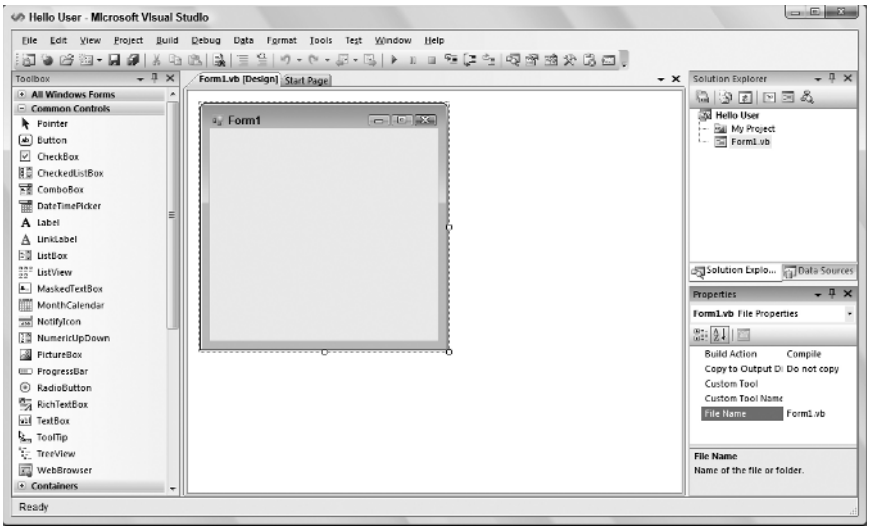

Figure 1-8

- **Toolbox:** The Toolbox contains reusable controls and components that can be added to your application. These range from buttons to data connectors to customized controls that you have either purchased or developed.
- **Design window:** The Design window is where a lot of the action takes place. This is where you will draw your user interface on your forms. This window is sometimes referred to as the Designer.
- **Solution Explorer:** The Solution Explorer window contains a hierarchical view of your solution. A *solution* can contain many projects, whereas a *project* contains forms, classes, modules, and components that solve a particular problem.
- **Data Sources:** The Data Sources window allows you to connect to a database and choose the database objects for your application.
- **Properties:** The Properties window shows what *properties* the selected object makes available. Although you can set these properties in your code, sometimes it is much easier to set them while you are designing your application (for example, drawing the controls on your form). You will notice that the `File Name` property has the value `Form1.vb`. This is the physical file name for the form's code and layout information.

Try It Out — Creating a HelloUser Project (cont.)

Next you'll give your form a name and set a few properties for it.

1. Change the name of your form to something more indicative of what your application is. Click `Form1.vb` in the Solution Explorer window. Then, in the Properties window, change the `File Name` property from `Form1.vb` to **HelloUser.vb** and press Enter, as shown in Figure 1-9. When changing properties you must either press Enter or click on another property for it to take effect.

Figure 1-9

2. Note that the form's file name has also been updated in the Solution Explorer to read `HelloUser.vb`.

3. Click the form displayed in the Design window. The Properties window will change to display the form's `Form` properties (instead of the `File` properties, which you have just been looking at). You will notice that the Properties window is dramatically different. The difference is the result of two different views of the same file. When the form name is highlighted in the Solution Explorer window, the physical file properties of the form are displayed. When the form in the Design window is highlighted, the visual properties and logical properties of the form are displayed.

   The Properties window allows you to set a control's properties easily. Properties are a particular object's set of internal data; they usually describe appearance or behavior. In Figure 1-10 you can see that properties are displayed alphabetically. The properties can also be grouped together in categories — Accessibility, Appearance, Behavior, Data, Design, Focus, Layout, Misc, and Window Style.

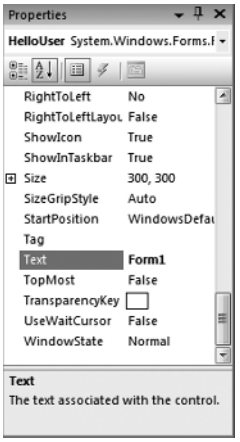

Figure 1-10

4. Right now, the title (`Text` property) of your form (displayed in the bar at the top) is `Form1`. This is not very descriptive, so change it to reflect the purpose of this application. Locate the `Text` property in the Properties window. Change the `Text` property's value to **Hello from Visual Basic 2008** and press Enter. Note that the form's title has been updated to reflect the change.

   If you have trouble finding properties, click the little AZ icon on the toolbar toward the top of the Properties window. This changes the property listing from being ordered by category to being ordered by name.

5. You are now finished with the procedure. Click the Start button on the Visual Studio 2008 toolbar (the green triangle) to run the application. As you work through the book, whenever we say "run the project" or "start the project," just click the Start button. An empty window with the title Hello from Visual Basic 2008 is displayed.

That was simple, but your little application isn't doing much at the moment. Let's make it a little more interactive. To do this, you are going to add some controls — a label, a text box, and two buttons to the form. This will let you see how the Toolbox makes adding functionality quite simple. You may be wondering at this point when you will actually look at some code. Soon! The great thing about Visual Basic 2008 is that you can develop a fair amount of your application *without* writing any code. Sure, the code is still there, behind the scenes, but, as you will see, Visual Basic 2008 writes a lot of it for you.

The Toolbox

The Toolbox is accessed through the View ⇨ Toolbox menu option, by clicking the Toolbox icon on the Standard menu bar, or by pressing Ctrl+Alt+X. Alternatively, the Toolbox tab is displayed on the left of the IDE; hovering your mouse over this tab will cause the Toolbox window to fly out, partially covering your form.

The Toolbox contains a Node type view of the various controls and components that can be placed onto your form. Controls such as text boxes, buttons, radio buttons, and combo boxes can be selected and then *drawn* onto your form. For the HelloUser application, you will be using only the controls in the Common Controls node. Figure 1-11 shows a listing of common controls for Windows Forms.

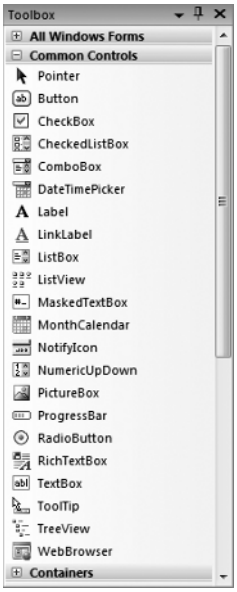

Figure 1-11

Controls can be added to your forms in any order, so it does not matter if you add the label control after the text box or the buttons before the label. In the following Try It Out exercise, you start adding controls.

Try It Out — Adding Controls to the HelloUser Application

1. Stop the project if it is still running, because you now want to add some controls to your form. The simplest way to stop your project is to click the close (X) button in the top-right corner of the form. Alternatively, you can click the blue square on the toolbar (which displays a ToolTip that says "Stop Debugging" if you hover over it with your mouse pointer).

2. Add a Label control to the form. Click Label in the Toolbox, drag it over to the form's Designer and drop it in the desired location. (You can also place controls on your form by double-clicking the required control in the Toolbox or clicking the control in the Toolbox and then drawing it on the form.)

3. If the Label control you have just drawn is not in the desired location, it really isn't a problem. When the control is on the form, you can resize it or move it around. Figure 1-12 shows what

the control looks like after you place it on the form. To move it, click the dotted border and drag it to the desired location. The label will automatically resize itself to fit the text that you enter in the `Text` property.

Figure 1-12

4. After drawing a control on the form, you should at least configure its name and the text that it will display. You will see that the Properties window to the right of the Designer has changed to `Label1`, telling you that you are currently examining the properties for the label. In the Properties window, set your new label's `Text` property to **Enter Your Name**. Note that, once you press Enter or click on another property, the label on the form has automatically resized itself to fit the text in the `Text` property. Now set the `Name` property to **lblName**.

5. Now, directly beneath the label, you want to add a text box, so that you can enter a name. You are going to repeat the procedure you followed for adding the label, but this time make sure you select the TextBox control from the toolbar. After you have dragged and dropped (or double-clicked) the control into the appropriate position as shown in Figure 1-13, use the Properties window to set its `Name` property to **txtName**.

Notice the sizing handles on the left and right side of the control. You can use these handles to resize the text box horizontally.

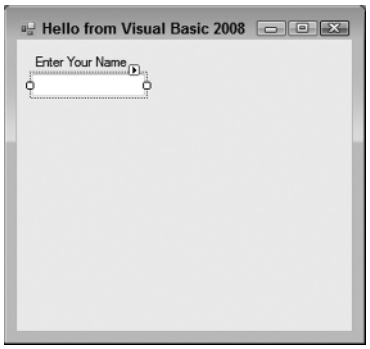

Figure 1-13

6. In the bottom left corner of the form, add a Button control in exactly the same manner as you added the label and text box. Set its `Name` property to **btnOK** and its `Text` property to **&OK**. Your form should now look similar to the one shown in Figure 1-14.

The ampersand (&) is used in the `Text` *property of buttons to create a keyboard shortcut (known as a* hot *key). The letter with the & sign placed in front of it will become underlined (as shown in Figure 1-14) to signal users that they can select that button by pressing the Alt+letter key combination, instead of using the mouse (on some configurations the underline doesn't appear until the user presses Alt). In this particular instance, pressing Alt+O would be the same as clicking the OK button. There is no need to write code to accomplish this.*

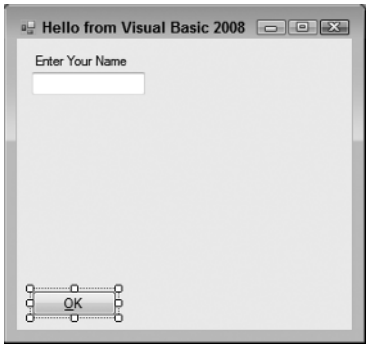

Figure 1-14

7. Now add a second Button control to the bottom right corner of the form by dragging the Button control from the Toolbox onto your form. Notice that, as you get close to the bottom right of the form, a blue snap line appears, as shown in Figure 1-15. This snap line allows you to align this new Button control with the existing Button control on the form. The snap lines assist you in aligning controls to the left, right, top, or bottom of each other, depending on where you are trying to position the new control. The light blue line provides you with a consistent margin between the edge of your control and the edge of the form. Set the `Name` property to **btnExit** and the `Text` property to **E&xit**. Your form should look similar to Figure 1-16.

Figure 1-15

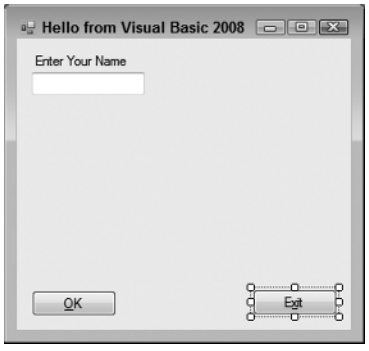

Figure 1-16

Now before you finish your sample application, let us briefly discuss some coding practices that you should be using.

Modified Hungarian Notation

You may have noticed that the names given to the controls look a little funny. Each name is prefixed with a shorthand identifier describing the type of control it is. This makes it much easier to understand what type of control you are working with when you are looking through the code. For example, say you had a control called simply Name, without a prefix of `lbl` or `txt`. You would not know whether you were working with a text box that accepted a name or with a label that displayed a name. Imagine if, in the previous Try It Out exercise, you had named your label Name1 and your text box Name2 — you would very quickly become confused. What if you left your application for a month or two and then came back to it to make some changes?

When working with other developers, it is very important to keep the coding style consistent. One of the most commonly used styles for control names within application development in many languages was designed by Dr. Charles Simonyi, who worked for the Xerox Palo Alto Research Center (XPARC) before joining Microsoft. He came up with short prefix mnemonics that allowed programmers to easily identify the type of information a variable might contain. Because Simonyi is from Hungary, and the prefixes make the names look a little foreign, the name *Hungarian Notation* came into use for this system. Because the original notation was used in C/C++ development, the notation for Visual Basic 2008 is termed Modified. Table 1-1 shows some of the commonly used prefixes that you will be using in this book.

Table 1-1: Common Prefixes in Visual Basic 2008

Control	Prefix
Button	`btn`
ComboBox	`cbo`
CheckBox	`chk`
Label	`lbl`
ListBox	`lst`
MainMenu	`mnu`
RadioButton	`rdb`
PictureBox	`pic`
TextBox	`txt`

Hungarian Notation can be a real time-saver when you are looking at code someone else wrote or at code that you wrote months earlier. However, by far the most important thing is to be consistent in your naming. When you start coding, choose a convention for your naming. It is recommended that you use the de facto standard Modified-Hungarian for Visual Basic 2008, but it is not required. After you pick a convention, stick to it. When modifying others' code, use theirs. A standard naming convention followed throughout a project will save countless hours when the application is maintained. Now let's get back to the application. It's now time to write some code.

The Code Editor

Now that you have the HelloUser form defined, you have to add some code to make it actually do something interesting. You have already seen how easy it is to add controls to a form. Providing the functionality behind those on-screen elements is no more difficult. To add the code for a control, you just double-click the control in question. This opens the code editor in the main window, shown in Figure 1-17.

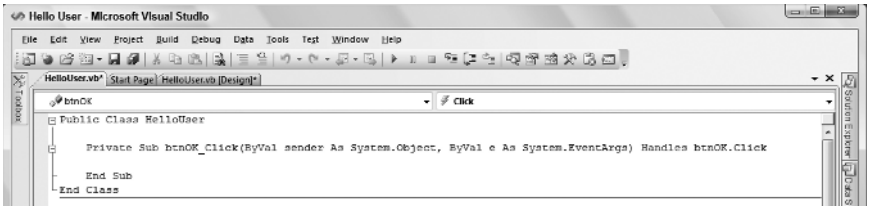
Hello User - Microsoft Visual Studio
File Edit View Project Build Debug Data Tools Test Window Help
HelloUser.vb* Start Page HelloUser.vb [Design]*
btnOK
Click
Public Class HelloUser
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnOK.Click
End Sub
End Class

Figure 1-17

Note that an additional tab has been created in the main window. Now you have the Design tab and the Code tab, each containing the name of the form you are working on. You draw the controls on your form in the Design tab, and you write code for your form in the Code tab. One thing to note here is that Visual Studio 2008 has created a separate file for the code. The visual definition and the code behind it exist in separate files: `HelloUser.Designer.vb` and `HelloUser.vb`. This is actually the reason why building applications with Visual Basic 2008 is so slick and easy. Using the Design view you can visually lay out your application, and then, using Code view, you add just the bits of code to implement your desired functionality.

Note also that there are two combo boxes at the top of the window. These provide shortcuts to the various parts of your code. Hover your mouse on the combo box on the left, and you'll see a tooltip appear, telling you that it is the Class Name combo box. If you expand this combo box, you will see a list of all the objects within your application. If you hover your mouse on the combo box on the right, you'll see a tooltip telling you that this is the Method Name combo box. If you expand this combo box, you will see a list of all defined functions and subroutines for the object selected in the Class Name combo box. If this particular form had a lot of code behind it, these combo boxes would make navigating to the desired code area very quick — jumping to the selected area in your code. However, since all of the code for this project so far fits in the window, there are not a lot of places to get lost.

Try It Out — Adding Code to the HelloUser Project

1. To begin adding the necessary code, click the Design tab to show the form again. Then double-click the OK button. The code window will open with the following code. This is the shell of the button's `Click` event and is the place where you enter the code that you want to run when you click the button. This code is known as an *event handler* and sometimes is also referred to as *an event procedure:*

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnOK.Click

End Sub
```

As a result of the typographic constraints in publishing, it is not possible to put the `Sub` *declaration on one line. Visual Basic 2008 allows you to break up lines of code by using the underscore character (_) to signify a line continuation. The space before the underscore is required. Any whitespace preceding the code on the following line is ignored.*

`Sub` is an example of a *keyword*. In programming terms, a keyword is a special word that is used to tell Visual Basic 2008 to do something special. In this case, it tells Visual Basic 2008 that this is a *subroutine*, a procedure that does not return a value. Anything that you type between the lines `Private Sub` and `End Sub` will make up the event procedure for the OK button.

2. Now add the highlighted code into the procedure:

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnOK.Click
    'Display a message box greeting to the user
    MessageBox.Show("Hello, " & txtName.Text & _
        "! Welcome to Visual Basic 2008.", _
        "Hello User Message")
End Sub
```

Throughout this book, you will be presented with code that you should enter into your program if you are following along. Usually, we will make it pretty obvious where you put the code, but as we go, we will explain anything that looks out of the ordinary. The code with the gray background is code that you should enter.

3. After you have added the preceding code, go back to the Design tab, and double-click the Exit button. Add the following highlighted code to the `btnExit_Click` event procedure:

```
Private Sub btnExit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click
    'End the program and close the form
    Me.Close()
End Sub
```

You may be wondering what `Me` is. `Me` is a keyword that refers to the form. Just like the pronoun *me*, it is just shorthand for referring to one's self.

4. Now that the code is finished, the moment of truth has arrived and you can see your creation. First, however, save your work by using File ⇨ Save All from the menu or by clicking the Save All button on the toolbar. The Save Project dialog box is displayed as shown in Figure 1-18, prompting you for a name and location for saving the project.

By default, a project is saved in a folder with the project name; in this case Hello User. Since this is the only project in the solution, there is no need to create a separate folder for the solution which contains the same name as the project, thus the Create directory for solution check box has been unchecked.

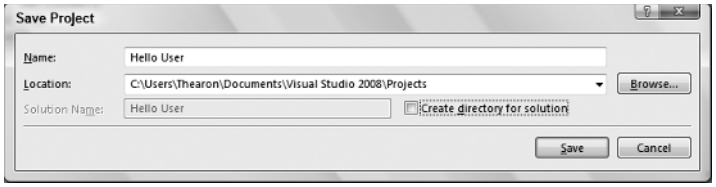

Figure 1-18

5. Now click the Start button on the toolbar. You will notice a lot of activity in the Output window at the bottom of your screen. Provided that you have not made any mistakes in entering the code, this information just lets you know which files are being loaded to run your application.

   At this point Visual Studio 2008 will *compile* the code. Compiling is the activity of taking the Visual Basic 2008 source code that you have written and translating it into a form that the computer understands. After the compilation is complete, Visual Studio 2008 runs (also known as *executes*) the program, and you'll be able to see the results.

   Any errors that Visual Basic 2008 encounters will be displayed as tasks in the Task List window. Double-clicking a task transports you to the offending line of code. You will learn more about how to debug the errors in your code in Chapter 10.

6. When the application loads, you see the main form. Enter a name and click OK or press the Alt+O key combination (see Figure 1-19).

Figure 1-19

A window known as a message box appears as shown in Figure 1-20, welcoming the person whose name was entered in the text box on the form — in this case Stephanie.

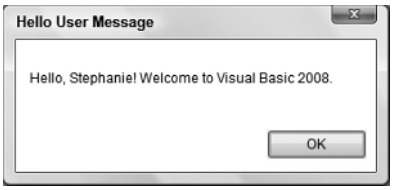

Figure 1-20

7. After you close the message box by clicking the OK button, click the Exit button on your form. The application closes and you will be returned to the Visual Basic 2008 IDE.

How It Works

The code that you added to the `Click` event for the OK button will take the name that was entered in the text box and use it as part of the message that was displayed in Figure 1-20.

The first line of text you entered in this procedure (`'Display a message box greeting to the user`) is actually a *comment*, text that is meant to be read by the human programmer who is writing or maintaining the code, not by the computer. Comments in Visual Basic 2008 begin with a single quote (`'`), and everything following on that line is considered a comment and ignored by the compiler. Comments will be discussed in detail in Chapter 3.

The `MessageBox.Show` method displays a message box that accepts various parameters. As used in your code, you have passed the string text to be displayed in the message box. This is accomplished through the *concatenation* of string constants defined by text enclosed in quotes. Concatenation of strings into one long string is performed through the use of the ampersand (`&`) character.

The code that follows concatenates a string constant of `"Hello, "` followed by the value contained in the `Text` property of the `txtName` text box control followed by a string constant of `"! Welcome to Visual Basic 2008."`. The second parameter being passed to the `MessageBox.Show` method is the caption to be used in the title bar of the Message Box dialog box.

Finally, the underscore (_) character used at the end of the lines in the following code enables you to split your code onto separate lines. This tells the compiler that the rest of the code for the parameter is continued on the next line. This is really useful when building long strings, because it allows you to view the entire code fragment in the Code Editor without having to scroll the Code Editor window to the right to view the entire line of code.

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnOK.Click

    'Display a message box greeting to the user
    MessageBox.Show("Hello," & txtName.Text & _
      "! Welcome to Visual Basic 2008.", _
      "Hello User Message")
End Sub
```

The next procedure that you added code for was the Exit button's `Click` event. Here you simply enter the code: `Me.Close()`. As explained earlier, the `Me` keyword refers to the form itself. The `Close` method of the form closes the form and releases all resources associated with it, thus ending the program.

```
Private Sub btnExit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click

    'End the program and close the form
    Me.Close()
End Sub
```

Using the Help System

The Help system included in Visual Basic 2008 is an improvement over the Help systems in earlier versions. As you begin to learn Visual Basic 2008, you will probably become very familiar with the Help system. However, it is worthwhile to give you an overview, just to help speed your searches for information.

The Help menu contains the items shown in Figure 1-21.

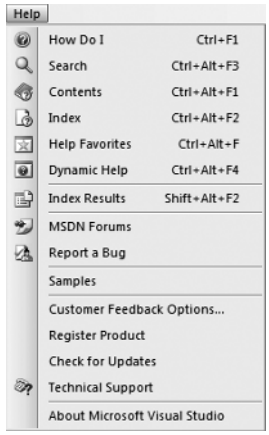

Figure 1-21

As you can see, this menu contains a few more items than the typical Windows application. The main reason for this is the vastness of the documentation. Few people could keep it all in their heads — but luckily, that is not a problem, because you can always quickly and easily refer to the Help system. Think of it as a safety net for your brain.

One really fantastic feature is Dynamic Help. When you select the Dynamic Help menu item from the Help menu, the Dynamic Help window is displayed as a tab behind the Properties window, with a list of relevant topics for whatever you may be working on.

Suppose you are working with a text box (perhaps the text box in the HelloUser application) and want to find out some information; you just select the text box on your form or in the code window and then use Dynamic Help to see all the help topics that pertain to text boxes, as shown in Figure 1-22.

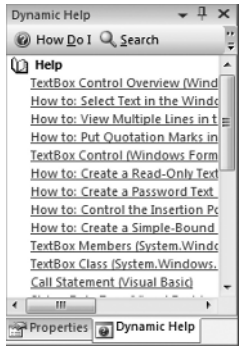

Figure 1-22

The other help commands in the Help menu (Search, Contents, and Index), function just as they would in any other Windows application. The How Do I menu item displays the Visual Studio Help collection with a list of common tasks that are categorized. This makes finding help on common tasks fast and efficient.

Summary

Hopefully, you are beginning to see that developing basic applications with Visual Basic 2008 is not that difficult. You have taken a look at the IDE and have seen how it can help you put together software very quickly. The Toolbox enables you to add controls to your form and design a user interface very quickly and easily. The Properties window makes configuring those controls a snap, while the Solution Explorer gives you a bird's eye view of the files that make up your project. You even wrote a little code.

In the coming chapters, you will go into even more detail and get comfortable writing code. Before you go too far into Visual Basic 2008 itself, the next chapter will give you an introduction to the Microsoft .NET Framework. This Framework is what gives all of the .NET languages their ease of use, ease of interoperability, and simplicity in learning.

To summarize, you should now be familiar with:

- ❑ The integrated development environment (IDE)
- ❑ Adding controls to your form in the Designer
- ❑ Setting the properties of your controls
- ❑ Adding code to your form in the code window

Exercise

The answers for this exercise and those at the end of each chapter in this book can be found in Appendix A.

1. Create a Windows Application with a Textbox and Button control that will display whatever is typed in the text box when the user clicks on the button.